CSE 413 Spring 1999

Java Streams

Streams

- Stream = flow of data (bytes or characters)
- Can be associated with files, communication links, keyboard/screen/printer
- Many stream classes; most are designed to be used as wrappers that accept data and transform or filter it before passing it along
- Java 1.0: Byte streams with a few wrappers to handle ASCII text
- Java 1.1: Added text stream classes to handle Unicode text properly

Stream Classes (1)

- **`InputStream/OutputStream`** - abstract classes defining basic raw byte stream operations
- **`Reader/Writer`** - abstract classes defining basic text stream operations

All Java stream classes are built on top of these

- **`InputStreamReader/OutputStreamWriter`** - basic conversion between bytes and characters (in both directions)

Stream Classes (2)

- **`BufferedInputStream/ BufferedOutputStream BufferedReader/BufferedWriter`** - versions of streams that add buffering and additional input/output methods
- **`PrintWriter`** - Text stream with methods for printing **`String`**s and primitive types as text output.

Stream Classes (3)

- **`DataInputStream/DataOutputStream`** - Filter streams that can read/write simple types including **`String`** and primitive numeric types as binary byte streams.
- **`FileInputStream/FileOutputStream FileReader/FileWriter`** - byte and text streams that read and write from/to the local file system.

Ex: Read a byte from Keyboard

- **`System.in`** is an **`InputStream`**. At the lowest level, we can read bytes. As in C, the basic **`read()`** operation returns an **`int`**, with -1 indicating end of stream.

```
try {
  int nibble = System.in.read();
} catch (IOException e) { … }
```

Ex: Read Line from Keyboard

- To read lines of characters, convert **`System.in`** to a character stream, and wrap it in a **`BufferedReader`** to get **`readLine()`**.

```
try {
  InputStreamReader chars =
   new InputStreamReader(System.in);
  BufferedReader in =
   new BufferedReader(chars);
  String firstLine = in.readLine();
  …
} catch (IOException e) { … }
```


Formatted I/O

- **`java.txt`** has many classes for formatting output and parsing input (new in Java 1.1).

```
NumberFormat nf =
   NumberFormat.getInstance();
for (double x = Math.PI;
       x < 100000; x = x*10) {
   String ns = nf.format(x);
   System.out.println(ns +
                      '\t' + x);
}
```


Output

```
3.141       3.14159265358979
31.415      31.4159265358979
314.159     314.159265358979
3,141.592   3141.59265358979
31,415.926  31415.9265358979
...
```

- Almost any formatting option you might want is available, and formatting is sensitive to the current language (locale) being used.

File I/O

- The file stream classes have constructors that take a filename as an argument and open the file.

```
Try {
  FileReader theFile =
     new FileReader("input.dat");
  BufferedReader input =
     new BufferedReader(theFile);
  String line = input.readLine();
  System.out.println(line);
} catch (IOException e) { … }
```

- Gotcha: File names depend on the underlying file system -- hard to be completely "platform independent".

Selecting Files

- Class FileDialog lets the user select the file with a dialog box

```
Try {
 FileDialog fd=new FileDialog(this,
     "Pick File",FileDialog.LOAD);
  fd.show();
  fileName = fd.getfile();
  if (filename != null) {
    // use fileName to open the file
     …
  }
} catch (IOException e) { … }
```